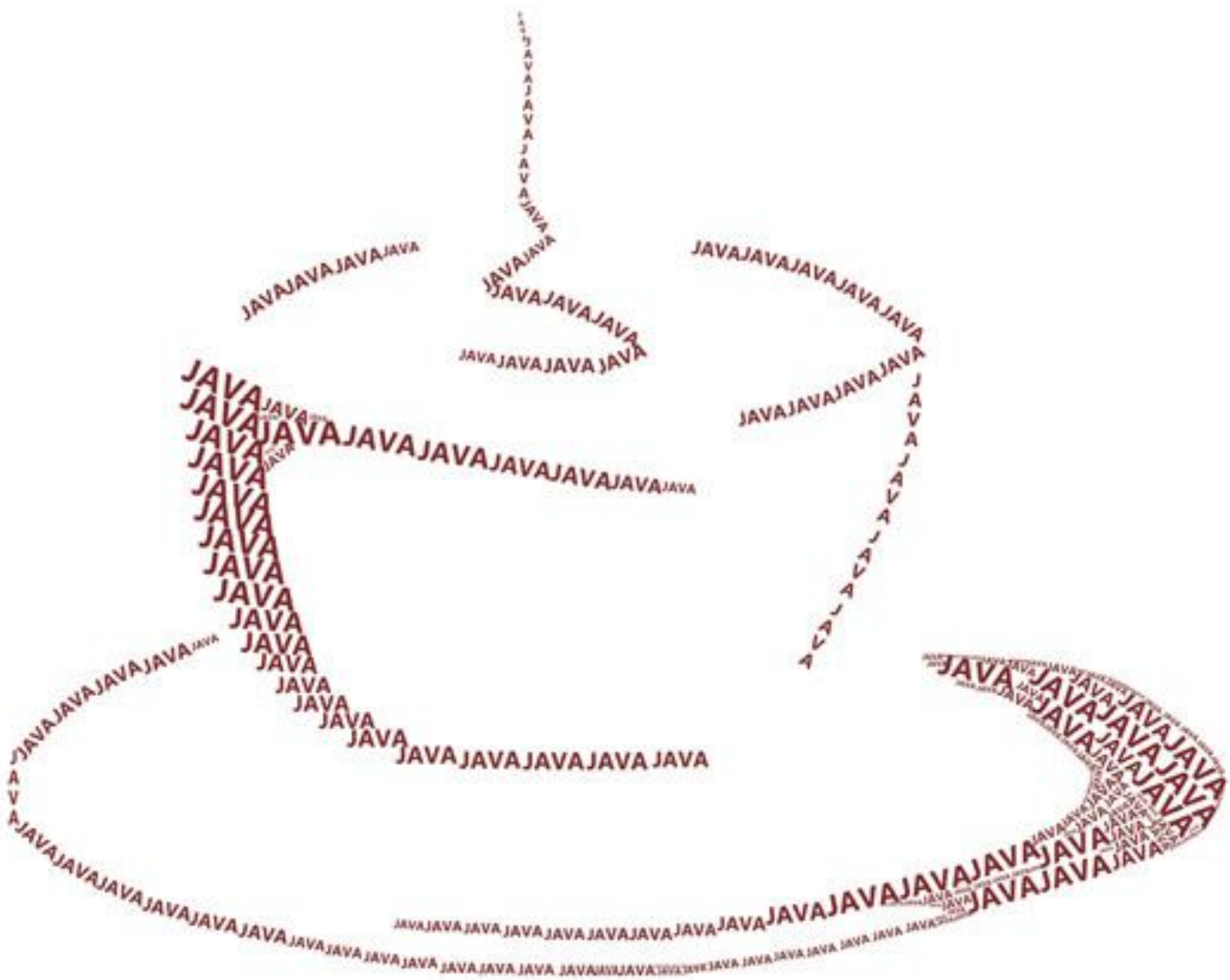
ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ Java

2η ΕΚΔΟΣΗ



Ι. Κ. ΚΑΒΟΥΡΑΣ
Α. Α. ΡΟΥΚΟΥΝΑΚΗ

ΚΛΕΙΔΑΡΙΘΜΟΣ

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Επιτέλους ανακάλυψα ότι όλα τα πράγματα είχαν ονόματα. Μετά άρχισα να διερωτώμαι γιατί είχαν τα ονόματα που είχαν. Όταν τελικά εγκατέλειψα την προσπάθεια αυτή, τότε άρχισα να αναγνωρίζω τον κόσμο και να προσπαθώ να επικοινωνήσω με τους άλλους με τις λέξεις που άκουγα και είχα μάθει απλώς να ξεχωρίζω.
jc

«Όταν χρησιμοποιώ μια λέξη», είπε ο Humpty Dumpty σε μάλλον χλευαστικό τόνο, «σημαίνει ό,τι εγώ την επιλέγω να σημαίνει· τίποτα περισσότερο ή λιγότερο».
***Lewis Carroll**, Μέσα από τον Καθρέφτη*

Θα τα κάναμε μούσκεμα αν τα κάναμε μαζί. Βλέπεις, είμαστε διαφορετικοί τύποι.

1.1. Οργάνωση υπολογιστών

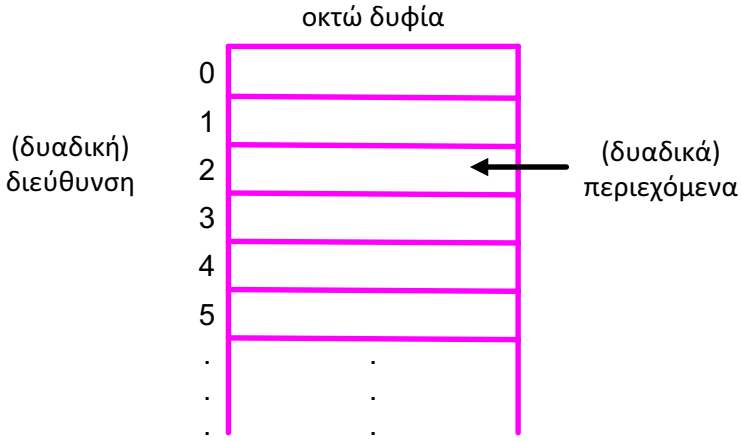
Όλοι οι υπολογιστές από πρώτης ως και τέταρτης γενιάς, εκτός από τους υπερυπολογιστές, αποτελούνται από τέσσερα μέρη:

- την κύρια μνήμη
- την Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ)
- τη βοηθητική μνήμη και
- τις συσκευές εισόδου/εξόδου.

Η **κύρια** ή **εσωτερική** ή **κεντρική μνήμη** (**main** ή **internal** ή **central memory**) χρησιμοποιείται για να αποθηκεύει τα προγράμματα που εκτελεί ο υπολογιστής, καθώς και τα δεδομένα που απαιτούνται και τα αποτελέσματα που παράγονται από την εκτέλεση των προγραμμάτων αυτών. Γι' αυτόν το λόγο η κύρια μνήμη είναι γνωστή και ως **κύρια αποθήκη** (**main store** ή **storage**). Επειδή η κύρια μνήμη έχει πεπερασμένη χωρητικότητα και είναι σχετικά ακριβή, χρησιμοποιείται για να αποθηκεύει εντολές και δεδομένα που χρειάζονται άμεσα. Τα υπόλοιπα αποθηκεύονται στη *μεγαλύτερη*, *φτηνότερη* και *πιο αργή* βοηθητική μνήμη ή αποθήκη (βλ. παρακάτω).

Η κύρια μνήμη αποτελείται από έναν αριθμό θέσεων ή κελιών, καθένα από τα οποία περιέχει το ίδιο πλήθος **δυαδικών ψηφίων** (**binary digits**), γνωστών ως **δυφίων** (**bits**). Σε κάθε θέση αντιστοιχεί μια (δυαδική) διεύθυνση, δηλαδή ένας αριθμός μέσω του οποίου ένα πρόγραμμα μπορεί να αναφερθεί στη συγκεκριμένη θέση. Κάθε θέση της μνήμης

είναι ικανή να αποθηκεύσει μια **δυφιοσυλλαβή** (**byte**), δηλαδή μια ομάδα οκτώ δυφίων, όπως δείχνει το Σχήμα 1.1.



Σχήμα 1.1. Κύρια μνήμη ή αποθήκη.

*Πρέπει να μπορούμε να διακρίνουμε τη **διεύθυνση** μιας θέσης της μνήμης από τα **περιεχόμενά** της.*

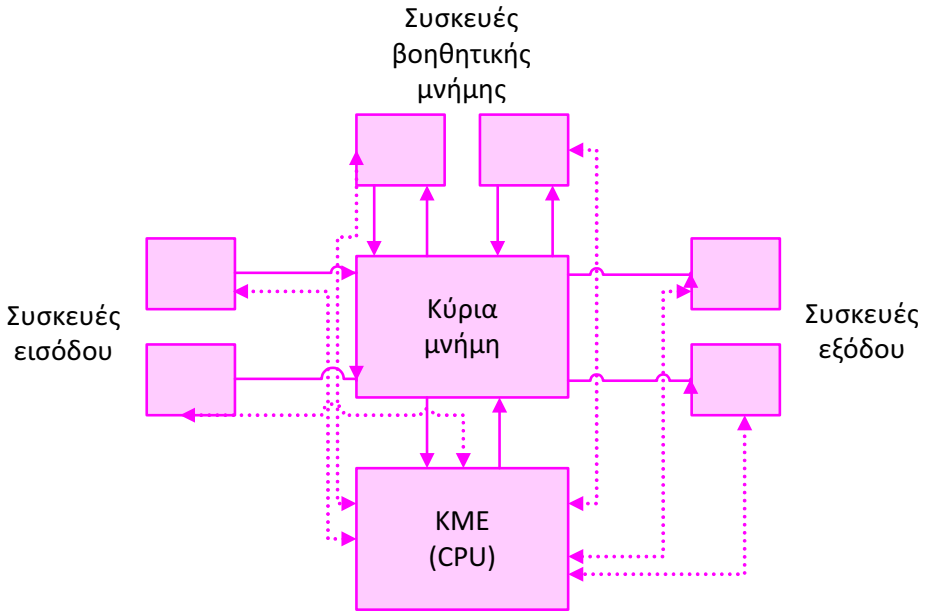
Η **Κεντρική Μονάδα Επεξεργασίας**, **ΚΜΕ** (**Central Processing Unit**, **CPU**), που είναι γνωστή και ως **κεντρικός επεξεργαστής** (**central processor**), ανακαλεί αυτόματα εντολές από την κύρια μνήμη και τις εκτελεί ακολουθιακά. Η ΚΜΕ περιέχει έναν αριθμό *καταχωρητών* (*registers*), στους οποίους τοποθετεί τα δεδομένα. Στη συνέχεια μπορεί να εκτελέσει απλές πράξεις σε αυτά, όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό κ.ά. Τα αποτελέσματα των πράξεων καταλήγουν και αυτά στους καταχωρητές. Οι καταχωρητές είναι παρόμοιοι με το *συσσωρευτή* (*accumulator*) που διαθέτουν οι υπολογιστές τσέπης.

*Ο **υπολογιστής** είναι ένας **επεξεργαστής** με (κύρια) **μνήμη**.*

Οι πληροφορίες μεταδίδονται από τον «εξωτερικό κόσμο» στον υπολογιστή μέσω των **συσκευών εισόδου** (**input devices**) και από τον υπολογιστή στον «εξωτερικό κόσμο» μέσω των **συσκευών εξόδου**. Υπάρχουν πολλές συσκευές εισόδου/εξόδου. Οι πιο συνηθισμένες συσκευές εισόδου είναι το *πληκτρολόγιο* και το *ποντίκι,* ενώ οι πιο συνηθισμένες συσκευές εξόδου είναι η *οθόνη* και ο *εκτυπωτής*. Τέλος, οι πιο γνωστές **συσκευές βοηθητικής αποθήκευσης** (**backing storage devices**) είναι οι *σκληροί δίσκοι*. Οι μονάδες εισόδου/εξόδου και η βοηθητική μνήμη είναι γνωστές ως **περιφερειακές συσκευές** ή **περιφερειακές μονάδες** (**peripheral devices** ή **peripheral units**).

Τα κυκλώματα και τα μηχανικά τμήματα ενός υπολογιστή αποτελούν το **υλισμικό** ή **υλικό** (**hardware**) του, το οποίο οργανώνεται όπως δείχνει το Σχήμα 1.2. Οι διακεκομμέ-

νες γραμμές δείχνουν τη φορά των *σημάτων ελέγχου* (*control signals*), ενώ οι συνεχόμενες τη *ροή των δεδομένων* (*data flow*). Ο όρος **σύστημα υπολογιστή** (**computer system**) περιλαμβάνει, εκτός από τον ίδιο τον υπολογιστή, τις περιφερειακές συσκευές του και όλα τα προγράμματα που ελέγχουν τη λειτουργία του και βοηθούν τους χρήστες να επικοινωνούν με αυτόν και να εκτελούν τις εργασίες τους. Τα προγράμματα αυτά αποτελούν το **λογισμικό** (**software**) του συστήματος[1].



Σχήμα 1.2. Οργάνωση υπολογιστών.

1.2. Αλγόριθμοι

Η δουλειά των υπολογιστών είναι να λύνουν προβλήματα ακολουθώντας τις δικές μας εντολές. Οι εντολές αυτές δεν πρέπει σε καμία περίπτωση να είναι *διφορούμενες* (*ambiguous*), δηλαδή δεν πρέπει να είναι δυνατό να ερμηνευτούν με περισσότερους από έναν τρόπους. Για παράδειγμα, μια εντολή της μορφής «υπολόγισε το μέσο όρο των βαθμών των φοιτητών», είναι πολύ ασαφής. Πάρα πολλές λεπτομέρειες έχουν παραλειφθεί, για παράδειγμα «ποιων φοιτητών», «σε ποιο ή σε ποια μαθήματα», «πού βρίσκονται οι βαθμοί», «πόσοι είναι οι βαθμοί», «θα ληφθούν υπόψη όσοι φοιτητές έδωσαν λευκή κόλλα» κ.ά.

[1] Για περισσότερες πληροφορίες βλ. Κάβουρα Ι. Κ. Οργάνωση Συστημάτων Υπολογιστών (Συστήματα Υπολογιστών, Τόμος Ι), εκδ. Κλειδάριθμος, Έκδοση 7η, Αθήνα 2007.

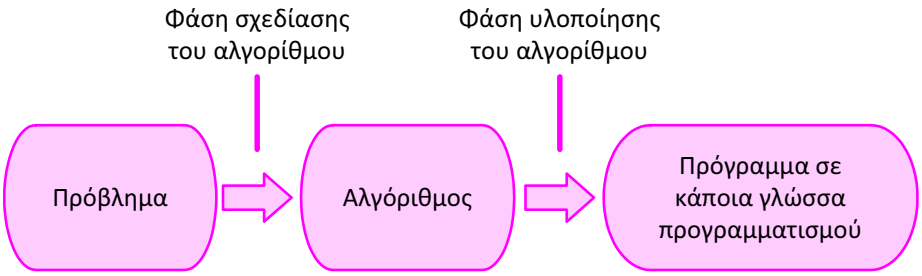
Υπάρχουν πάρα πολλά προβλήματα που φαίνονται δύσκολα ή πολύπλοκα από τη σκοπιά του προγραμματισμού. Η πολυπλοκότητα κάποιων από αυτά (π.χ. κάποιων σύνθετων μαθηματικών προβλημάτων) μπορεί να οφείλεται στην ίδια τη φύση τους, ενώ κάποιων άλλων (π.χ. του παραπάνω προβλήματος υπολογισμού του μέσου όρου των βαθμών των φοιτητών) μπορεί να οφείλεται απλώς στην έλλειψη σαφήνειας κατά την περιγραφή τους.

Βασικό ρόλο στην επίλυση των προβλημάτων παίζουν οι *αλγόριθμοι*. Ο **αλγόριθμος** (**algorithm**)[2] είναι μια διατεταγμένη ακολουθία βημάτων που οδηγεί στη λύση ενός προβλήματος. Η σχεδίαση αλγορίθμων είναι μια δημιουργική εργασία και μπορεί να θεωρηθεί μια μορφή τέχνης. Γι' αυτό, όπως συμβαίνει και με την τέχνη, υπάρχουν καλοί και κακοί αλγόριθμοι. Τα βασικά χαρακτηριστικά ενός καλού αλγορίθμου είναι:

- η **ακρίβεια** (**precision**): Τα βήματα του αλγορίθμου πρέπει να είναι σαφή και *μη διφορούμενα* (*unambiguous*).
- η **γενικότητα** (**generality**): Ο αλγόριθμος πρέπει να μπορεί να εφαρμοστεί σε πολλά και διαφορετικά δεδομένα.
- η **απόδοση** (**efficiency**): Ο αλγόριθμος θα πρέπει να απαιτεί όσο το δυνατόν λιγότερο χώρο και χρόνο για να εκτελεστεί.
- η **ορθότητα** (**correctness**): Ο αλγόριθμος πρέπει όντως να λύνει το πρόβλημα για το οποίο έχει σχεδιαστεί.
- ο **τερματισμός** (**termination**): Ο αλγόριθμος πρέπει να τερματίζεται μετά από την εκτέλεση ενός πεπερασμένου αριθμού βημάτων.

Κάθε φορά που θέλουμε να λύσουμε ένα πρόβλημα, είναι βασικό να μπορούμε να ξεχωρίζουμε τις δύο φάσεις που φαίνονται στο Σχήμα 1.3: τη *φάση σχεδίασης* ενός αλγορίθμου για το συγκεκριμένο πρόβλημα και τη *φάση υλοποίησης* του αλγορίθμου αυτού σε κάποια γλώσσα προγραμματισμού. Στη **φάση σχεδίασης** (**design phase**) του αλγορίθμου επικεντρώνουμε την προσοχή μας στην εύρεση ενός καλού αλγορίθμου για την επίλυση του προβλήματος. Όταν βεβαιωθούμε ότι ο αλγόριθμος που βρήκαμε είναι κατάλληλος για να λύσει το πρόβλημα, τότε μόνο στρέφουμε την προσοχή μας στη **φάση υλοποίησης** (**implementation phase**) του αλγορίθμου αυτού σε κάποια γλώσσα προγραμματισμού. Παρόλο που στο βιβλίο αυτό θα χρησιμοποιήσουμε τη γλώσσα Java, η περιγραφή των αλγορίθμων μας θα είναι τέτοια ώστε να μπορούν να υλοποιηθούν εύκολα σε οποιαδήποτε γλώσσα προγραμματισμού.

[2] Ο όρος προέρχεται από τη λατινική έκδοση του ονόματος του Πέρση μαθηματικού Al-Khwarizmi που έζησε γύρω στο 825 μ.Χ.



Σχήμα 1.3. Φάσεις επίλυσης ενός προβλήματος.

Ας δοκιμάσουμε να σχεδιάσουμε έναν αλγόριθμο που να εκτελεί μια γνωστή διαδικασία: την αντικατάσταση μιας καμένης λάμπας. Η βασική λειτουργία μπορεί να διαιρεθεί αρχικά σε δύο βήματα ως εξής:

```
Αφαίρεσε την καμένη λάμπα
Βάλε την καινούρια λάμπα
```

Τα βήματα αυτά φαίνεται να λύνουν το πρόβλημα, αλλά ας υποθέσουμε ότι οι οδηγίες μας προορίζονται για ένα οικιακό ρομπότ και ότι αυτές δεν είναι αρκετά απλές για να τις καταλάβει το ρομπότ μας που δεν γνωρίζει πολλές βασικές λεπτομέρειες. Έτσι προχωρούμε στον προσδιορισμό περισσότερων λεπτομερειών για κάθε βήμα.

Αν υποθέσουμε ότι η καμένη λάμπα βρίσκεται σε ένα πολύφωτο που είναι κρεμασμένο από το ταβάνι, τότε χρειαζόμαστε μια σκάλα. Αφού τοποθετήσουμε τη σκάλα κάτω από το πολύφωτο, πρέπει να ανεβούμε στη σκάλα και να γυρίσουμε τη λάμπα (*καλός αλγόριθμος*) ή να γυρίσουμε τη σκάλα (*κακός αλγόριθμος*) με αντιωρολογιακή φορά. Έτσι το πρώτο από τα αρχικά βήματα (`Αφαίρεσε την καμένη λάμπα`) αναλύεται στα εξής βήματα:

```
Βάλε τη σκάλα κάτω από την καμένη λάμπα
Ανέβα στη σκάλα
Όσο η καμένη λάμπα δεν αποσπάται
  Γύρνα τη με αντιωρολογιακή φορά
```

Το δεύτερο από τα αρχικά βήματα απαιτεί την επιλογή μιας λάμπας παρόμοιας με αυτήν που κάηκε (αν υπάρχει) και την τοποθέτηση της λάμπας αυτής στη θέση του πολύφωτου από την οποία αφαιρέσαμε την καμένη λάμπα. Έτσι, το βήμα `Βάλε την καινούρια λάμπα` αναλύεται στα εξής βήματα:

```
Διάλεξε μια καινούρια λάμπα ίδια με αυτή που κάηκε
Βάλε την καινούρια λάμπα στη σωστή θέση
Όσο η καινούρια λάμπα δεν βιδώνεται
  Γύρνα τη με ωρολογιακή φορά
```

Είναι ίσως φανερό ότι η επιλογή της καινούριας λάμπας πρέπει να προηγηθεί του βήματος `Ανέβα στη σκάλα` (*καλός αλγόριθμος*), γιατί στην αντίθετη περίπτωση το ρομπότ θα πρέπει να κατέβει και να ξανανέβει τη σκάλα (*κακός αλγόριθμος*). Είναι ίσως επίσης φανερό ότι το ρομπότ πρέπει να κατεβεί τελικά από τη σκάλα και να τη βάλει στη θέση της. Έτσι, στην τελική του μορφή ο αλγόριθμός μας είναι ο εξής:

```
Διάλεξε μια καινούρια λάμπα ίδια με αυτή που κάηκε
Αν βρήκες μια τέτοια λάμπα, τότε
  Βάλε τη σκάλα κάτω από την καμένη λάμπα
  Ανέβα στη σκάλα
  Όσο η καμένη λάμπα δεν αποσπάται
     Γύρνα τη με αντιωρολογιακή φορά
  Βάλε την καινούρια λάμπα στη σωστή θέση
  Όσο η καινούρια λάμπα δεν βιδώνεται
     Γύρνα τη με ωρολογιακή φορά
  Κατέβα από τη σκάλα
  Βάλε τη σκάλα στη θέση της
Ειδάλλως μην κάνεις τίποτα
```

Το πόσο λεπτομερής θα είναι ένας αλγόριθμος εξαρτάται πάντα από το ποιος θα τον εκτελέσει. Για παράδειγμα, μια συνταγή για κολοκυθόπιτα πρέπει να περιέχει πολύ περισσότερες λεπτομέρειες όταν προορίζεται για κάποιον που δεν ξέρει να βράζει ούτε ένα αυγό, ενώ μπορεί να είναι πολύ πιο σύντομη όταν προορίζεται για τον αρχιμάγειρα ενός μεγάλου ξενοδοχείου. Το ίδιο ισχύει και για τους αλγορίθμους που πρόκειται να εκτελεστούν από ένα υπολογιστικό σύστημα: Οι γλώσσες προγραμματισμού έχουν γενικά πολύ περιορισμένο λεξιλόγιο («ρεπερτόριο» εντολών) και οι αλγόριθμοι πρέπει τελικά να εκφραστούν σε αυτό.

Ασκήσεις 1.1.*

Κατά τη σχεδίαση του αλγορίθμου για την αντικατάσταση της καμένης λάμπας είδαμε δύο βασικές έννοιες: την έννοια της επανάληψης (Όσο) και την έννοια της λογικής απόφασης (Αν ... τότε ... ειδάλλως ...). Με βάση αυτές τις δύο (μόνο) έννοιες θα πρέπει τώρα οπωσδήποτε να επιχειρήσετε τουλάχιστον δύο από τις παρακάτω ασκήσεις, υποθέτοντας ότι εσείς είστε η (ανθρώπινη) μηχανή που θα εκτελέσει τους αλγορίθμους που θα σχεδιάσετε.

1. Σχεδιάστε έναν αλγόριθμο για την αντικατάσταση μιας ρόδας αυτοκινήτου. Υποθέστε ότι το αυτοκίνητό σας διαθέτει μια καλή ρεζέρβα και έναν κατάλληλο γρύλο.
2. Σχεδιάστε έναν αλγόριθμο για το ψήσιμο αβγών με μπέικον.
3. Σχεδιάστε έναν αλγόριθμο για την αντικατάσταση ενός σπασμένου τζαμιού σε κάποιο από τα παράθυρα του σπιτιού σας. Περιγράψτε ξεχωριστά σε έναν κατάλογο τα υλικά που θα χρειαστούν.

4. Σχεδιάστε έναν αλγόριθμο που να σας βγάζει το πρωί από το σπίτι σας. Το πρώτο βήμα πρέπει να είναι το `Άκουσα το ξυπνητήρι` και πρέπει να συμπεριλάβετε όλες τις συνηθισμένες σας ενέργειες.
5. Σχεδιάστε έναν αλγόριθμο για το πλύσιμο, το άπλωμα και το σιδέρωμα ρούχων.

1.3. Προγράμματα

Ας δούμε το πρώτο μας πρόγραμμα στη γλώσσα προγραμματισμού Java, το οποίο απλώς εμφανίζει στην οθόνη τη φράση `Γεια σου!`.

Παράδειγμα 1.1.

```
1  // GeiaSou.java
2  // Εμφανίζει τη φράση Γεια σου!.
3  public class GeiaSou
4  {
5      public static void main(String args[])
6      {
7          System.out.print("Γεια σου!"); /* αυτό θα γίνει */
8          // αυτό όχι
9          // System.out.print("Γεια σου και πάλι!");
10         System.out.println();
11     }
12 }
```

Οι γραμμές 1-2 του παραπάνω προγράμματος περιέχουν σχόλια. Στην Java υπάρχουν δύο ειδών σχόλια: τα *σχόλια υλοποίησης* και τα *σχόλια τεκμηρίωσης*. Τα **σχόλια υλοποίησης** (**implementation comments**) είτε ξεκινούν με `//`, οπότε επεκτείνονται μόνο μέχρι το τέλος της αντίστοιχης γραμμής, είτε περικλείονται μεταξύ των `/*` και `*/`, οπότε μπορούν να καταλαμβάνουν περισσότερες από μια γραμμές. Σκοπός τους είναι συνήθως να επεξηγήσουν το κομμάτι κώδικα που ακολουθεί μετά από αυτά (γραμμές 1, 2 και 8) ή το κομμάτι κώδικα που προηγείται (αν βρίσκονται στην ίδια γραμμή, όπως στη γραμμή 7), ή απλώς να ζητήσουν την παράβλεψη συγκεκριμένων κομματιών κώδικα από το μεταγλωττιστή (γραμμή 9).

Τα **σχόλια τεκμηρίωσης** (**documentation comments**) περικλείονται μεταξύ των συμβόλων `/**` και `*/`. Τα σχόλια αυτά περιέχουν συνήθως γενικές πληροφορίες για το πρόγραμμα. Χρησιμοποιώντας το εργαλείο `javadoc` μπορούμε να εξαγάγουμε τα σχόλια τεκμηρίωσης ενός προγράμματος σε αρχεία HTML, τα οποία στη συνέχεια μπορούν να χρησιμοποιηθούν από οποιονδήποτε που επιθυμεί να πάρει μια γενική ιδέα για το τι κάνει το πρόγραμμα, χωρίς να έχει στη διάθεσή του τον κώδικα του προγράμματος αυτού. Ένα σχόλιο τεκμηρίωσης για το παραπάνω πρόγραμμα θα μπορούσε να ήταν το παρακάτω:

```
/**
 * Εμφανίζει τη φράση Γεια σου!.
 * Γράφτηκε στις 01/09/2008.
 *
 * @author jc
 * @version 1.0
*/
```

Επειδή τα `//`, `/*`, `/**` και `*/` στην πραγματικότητα αντιπροσωπεύουν ένα μόνο σύμβολο, δεν πρέπει να υπάρχει κανένα κενό διάστημα μεταξύ των πλάγιων γραμμών και των αστερίσκων. Κατά σύμβαση, όλες οι γραμμές μέσα σε σχόλια που περικλείονται σε `/**` και `*/` ή σε `/*` και `*/` ξεκινούν με έναν αστερίσκο (που αγνοείται από τα εργαλεία τεκμηρίωσης), προκειμένου να διευκολύνεται η ανάγνωσή τους από το χρήστη.

Κάθε πρόγραμμα της Java χτίζεται από *τάξεις*. Οι γραμμές 3-12 του παραπάνω προγράμματος δηλώνουν μια *τάξη* (*class*) που ονομάζεται `GeiaSou`. Το τι είναι τάξη θα το εξηγήσουμε λεπτομερέστερα στην Κεφάλαιο 8. Προς το παρόν αρκεί να αναφέρουμε ότι:

- Κάθε πρόγραμμα της Java πρέπει να περιέχει τουλάχιστον μία τάξη.
- Η τάξη αποτελείται από *μέλη* (*members*) που δηλώνονται ανάμεσα στα άγκιστρα `{` και `}` που ακολουθούν το όνομά της.

Στην Επιστήμη των Υπολογιστών, τα *ονόματα* ονομάζονται **αναγνωριστικά** (**identifiers**).

*Στην Java τα **αναγνωριστικά** πρέπει να αρχίζουν είτε με κάποιο γράμμα του κώδικα UNICODE, είτε με χαρακτήρα υπογράμμισης (_), είτε με σύμβολο δολαρίου ($). Οι επόμενοι χαρακτήρες των αναγνωριστικών μπορούν να περιλαμβάνουν επίσης ένα ή περισσότερα αριθμητικά ψηφία (0-9).*

Οι *λέξεις-κλειδιά* (*keywords*) της Java (βλ. Παράρτημα Α), καθώς και οι σταθερές τιμές `null`, `false` και `true`, δεν μπορούν να χρησιμοποιηθούν ως αναγνωριστικά. Τέλος, είναι καλό να περιορίζουμε τη χρήση του χαρακτήρα δολαρίου (`$`) σε μηχανικά δημιουργούμενο κώδικα ή σε περιπτώσεις όπου θέλουμε να προσπελάσουμε προϋπάρχοντα ονόματα σε κληρονομημένα συστήματα (legacy systems).

Όλα τα αναγνωριστικά πρέπει να επιλέγονται από τους προγραμματιστές με τέτοιον τρόπο ώστε να δηλώνουν τη σημασία του αφηρημένου αντικειμένου που αντιπροσωπεύουν. Έτσι, αν χρειαζόταν να διαλέξουμε το αναγνωριστικό της τάξης του παραπάνω προγράμματος ανάμεσα στα:

```
Salute      Πρώτη      ΓειαΣου     Prog1      P001
```

είναι μάλλον φανερό ότι η επιλογή του ονόματος `Salute` ή του `ΓειαΣου` θα ήταν προτιμότερη από την επιλογή κάποιου από τα `Πρώτη`, `Prog1`, ή `P001`.

*Κατά σύμβαση, τα **αναγνωριστικά τάξεων** [και γενικά τύπων αναφορών (βλ. Κεφ. 8.1)] της Java είναι ουσιαστικά. Το πρώτο γράμμα ενός τέτοιου αναγνωριστικού, καθώς και το πρώτο γράμμα κάθε λέξης που περιέχεται σε αυτό είναι κεφαλαία. Όλα τα υπόλοιπα γράμματα του αναγνωριστικού είναι πεζά, με εξαίρεση την περίπτωση των ακρωνυμίων που μπορεί να περιέχονται μέσα σε ένα αναγνωριστικό και τα οποία γράφονται με κεφαλαία.*

Η τάξη `GeiaSou` περιέχει ένα μόνο μέλος, μια μέθοδο που ονομάζεται `main`. Η *δήλωση* μιας *μεθόδου* (*method declaration*) αποτελείται από την *επικεφαλίδα* και το *σώμα* της. Η *επικεφαλίδα*[3] της *μεθόδου* (*method header*) `main` πρέπει πάντα να είναι η:

```
public static void main(String args[])
```

ή η:

```
public static void main(String... args)
```

Η μέθοδος `main` πρέπει πάντα να δηλώνεται ως *δημόσια* και *στατική*. *Δημόσια μέθοδος* (*public method*) μιας τάξης είναι μια μέθοδος που μπορεί να χρησιμοποιηθεί (να κληθεί) από οποιαδήποτε μέθοδο μιας οποιασδήποτε άλλης τάξης που έχει πρόσβαση στην τάξη στην οποία έχει δηλωθεί η πρώτη (βλ. Εν. 8.1). *Στατική μέθοδος* (*static method*) είναι μια μέθοδος που ανήκει στην τάξη στην οποία αυτή έχει δηλωθεί, και όχι στα αντικείμενα της τάξης αυτής (βλ. Εν. 8.1.2). Τέλος, πριν από το όνομα της μεθόδου εμφανίζεται ο τύπος του αποτελέσματός της, που στην περίπτωση της `main` είναι `void` (κενός), επειδή η μέθοδος αυτή δεν επιστρέφει καμία τιμή.

Μετά το όνομα της μεθόδου ακολουθούν οι *παράμετροί της* (*parameters*), μία (πιθανόν κενή) ακολουθία από ζευγάρια τύπων και αναγνωριστικών, που ξεχωρίζονται με κόμματα και περικλείονται μέσα σε παρενθέσεις `(` και `)`. Οι παράμετροι μιας μεθόδου προσδιορίζουν από πού παίρνει τα δεδομένα της. Η μέθοδος `main` έχει μία μόνο παράμετρο: έναν *πίνακα* αντικειμένων τύπου `String` που ονομάζεται `args`. Οι *πίνακες* (*arrays*) συμβολίζονται με ένα ζεύγος από αγκύλες `[` και `]` μετά τον τύπο ή μετά από το αναγνωριστικό τους (βλ. Εν. 10.2). Ο πίνακας `args` είναι ο μηχανισμός μέσω του οποίου καθίσταται δυνατό να μεταβιβάζονται, κατά τη διάρκεια της εκτέλεσης του προγράμματος, πληροφορίες σε αυτό. Κάθε συμβολοσειρά του πίνακα `args` ονομάζεται **όρισμα της γραμμής εντολών** (**command-line argument**) (βλ. και Κεφ. 2.3.2).

Μετά την επικεφαλίδα μιας μεθόδου ακολουθεί το *σώμα* (*body*) της, που περιλαμβάνει *τις* εντολές *της*. Η κάθε εντολή πρέπει να τελειώνει με ένα ελληνικό ερωτηματικό (`;`). Το σώμα μιας μεθόδου αρχίζει με το ειδικό σύμβολο `{` (το οποίο σημαίνει «άρχισε») και τελειώνει με το ειδικό σύμβολο `}` (το οποίο σημαίνει «τέλειωσε») με εξαίρεση την περίπτωση που η μέθοδος αυτή είναι *αφηρημένη* (*abstract*) (βλ. Εν. 11.3).

[3] Στην ελληνική βιβλιογραφία αναφέρεται επίσης συχνά και ως *κεφαλίδα*.

Η μέθοδος `main` είναι μια ειδική μέθοδος: Πρέπει να δηλωθεί όπως ακριβώς περιγράψαμε και οι εντολές της εκτελούνται ακολουθιακά, αρχίζοντας από την πρώτη και τελειώνοντας στην τελευταία, όταν η τάξη που την περιέχει φορτώνεται (loaded), συνδέεται (linked) και αρχικοποιείται (initialized). Στο παράδειγμά μας υπάρχουν μόνο δύο εντολές μέσα στη μέθοδο `main`. Η εντολή:

```
System.out.print("Γεια σου!");
```

είναι μια κλήση της μεθόδου `print` του στατικού πεδίου `out` της τάξης `System` της Java με (πραγματικό) όρισμα (βλ. Εν. 7.2) τη συμβολοσειρά `"Γεια σου!"`. Η μέθοδος αυτή εμφανίζει στην οθόνη ό,τι βρίσκεται ανάμεσα στα διπλά εισαγωγικά " και ", *όπως ακριβώς αυτό έχει πληκτρολογηθεί* (οι χαρακτήρες λευκών διαστημάτων δεν αγνοούνται μέσα στις συμβολοσειρές). Η επόμενη εντολή του προγράμματος:

```
System.out.println();
```

αλλάζει γραμμή. Η μέθοδός μας θα μπορούσε να περιέχει μία μόνο εντολή, την:

```
System.out.println("Γεια σου!");
```

που κάνει και τις δύο δουλειές μαζί.

1.4. Μεταγλώττιση και εκτέλεση προγραμμάτων Java

Ένα *πρόγραμμα πηγής* (*source program*), ή *πηγαίο πρόγραμμα*, της Java μπορεί να γραφτεί σε κάποια εφαρμογή *συντάκτη* (*editor*) και πρέπει να αποθηκευτεί σε ένα αρχείο που να έχει το κατάλληλο όνομα τάξης (βλ. Εν. 12.6) και προέκταση `.java`. Αφού ετοιμάσουμε το πρόγραμμα πηγής, χρειάζεται να το μεταγλωττίσουμε στη γλώσσα της **Εικονικής Μηχανής Java** (**Java Virtual Machine, JVM**). Η γλώσσα αυτή ονομάζεται **κώδικας δυφιοσυλλαβών Java** (**Java byte code**).

Για να μεταγλωττίσουμε ένα πρόγραμμα σε κώδικα δυφιοσυλλαβών Java, χρησιμοποιούμε μια εντολή της μορφής:

```
javac όνομα_αρχείου.java
```

όπου ο `javac`[4] είναι ο **μεταγλωττιστής Java** (**Java compiler**) που παρέχεται από την εταιρεία Oracle. Το εργαλείο αυτό διαβάζει προγράμματα Java και τα μετατρέπει στα κατάλληλα κομμάτια κώδικα δυφιοσυλλαβών Java.

Για παράδειγμα, για να μεταγλωττίσουμε το πρόγραμμα του Παραδείγματος 1.1, πρέπει αρχικά να το γράψουμε και να το αποθηκεύσουμε σε ένα αρχείο με το όνομα `GeiaSou.java` και κατόπιν να εκτελέσουμε την εντολή:

[4] Αρχικά ο μόνος μεταγλωττιστής προγραμμάτων Java σε κώδικα δυφιοσυλλαβών Java ήταν ο `javac` της Oracle. Από τη στιγμή, όμως, που δημοσιοποιήθηκαν οι προδιαγραφές του κώδικα δυφιοσυλλαβών Java, έκαναν την εμφάνισή τους και άλλοι μεταγλωττιστές, όπως ο `gcj`, ο `jikes`, και ο `ecj`.

```
javac GeiaSou.java
```

Αν η μεταγλώττιση είναι επιτυχής, τότε ο μεταγλωττιστής θα παραγάγει ένα αρχείο με όνομα `όνομα_τάξης.class`, *για την κάθε τάξη που περιέχεται στο αρχείο πηγής*. Για να εκτελέσουμε το πρόγραμμα θα πρέπει να πληκτρολογήσουμε μια εντολή της μορφής:

```
java όνομα_τάξης όρισμα1 όρισμα2 ...
```

όπου ο `java` είναι ο **εκκινητής εφαρμογών Java** (**Java application launcher**). Το εργαλείο αυτό εκτελεί μια εφαρμογή της Java: Ξεκινά ένα **Περιβάλλον Εκτέλεσης Java** (**Java Runtime Environment**, **JRE**), φορτώνει μια συγκεκριμένη τάξη, και καλεί τη μέθοδο `main` της τάξης αυτής. Το `όνομα_τάξης` είναι το όνομα της τάξης που περιέχει τη μέθοδο `main`, της τάξης δηλαδή που φορτώνεται (μεταξύ άλλων) από τον εκκινητή.

Η εφαρμογή της Java μπορεί να δεχτεί ορίσματα από τη γραμμή εντολών. Όπως είδαμε παραπάνω, τα ορίσματα αυτά ακολουθούν το όνομα της τάξης στην εντολή εκτέλεσης της εφαρμογής. Κατά την εκκίνηση της εφαρμογής, το περιβάλλον εκτέλεσης μεταβιβάζει τα ορίσματα αυτά στη μέθοδο `main` μέσω του πίνακα `args`.

Έτσι, αν είναι επιτυχής η μεταγλώττιση του προγράμματος που βρίσκεται αποθηκευμένο στο αρχείο `GeiaSou.java`, θα δημιουργηθεί το αρχείο `GeiaSou.class`, το οποίο στη συνέχεια μπορούμε να το εκτελέσουμε με χρήση της εντολής:

```
java GeiaSou
```

Η εκτέλεση του προγράμματος αυτού θα εμφανίσει στην οθόνη μας το μήνυμα:

```
Γεια σου!
```

Παράδειγμα 1.2.

Είναι μάλλον φανερό πως, αν θέλαμε να πάρουμε στην οθόνη μας τα μηνύματα:

```
Καλή σου μέρα!
Πώς είσαι; Είσαι καλά;
```

θα έπρεπε να ετοιμάσουμε, να μεταγλωττίσουμε και να εκτελέσουμε το εξής πρόγραμμα.

```
1  // Xairetismos.java
2  // Εμφανίζει έναν τυπικό χαιρετισμό.
3  public class Xairetismos
4  {
5      public static void main(String arguments[])
6      {
7          System.out.println("Καλή σου μέρα!");
8          System.out.print("Πώς είσαι;");
9          System.out.println(" Είσαι καλά;");
10     }
11 }
```

Παράδειγμα 1.3.

Το παρακάτω πρόγραμμα εμφανίζει σε ξεχωριστές γραμμές της οθόνης τα τρία πρώτα ορίσματα που δίνονται στη γραμμή εντολών.

```
1  // Orismata.java
2  // Εμφανίζει τα τρία πρώτα ορίσματα της γραμμής εντολών.
3  public class Orismata
4  {
5      public static void main(String[] args)
6      {
7          System.out.println(args[0]);     // πρώτο όρισμα
8          System.out.println(args[1]);     // δεύτερο όρισμα
9          System.out.println(args[2]);     // τρίτο όρισμα
10     }
11 }
```

Αφού ετοιμάσουμε το πρόγραμμα αυτό, το αποθηκεύσουμε σε ένα αρχείο με όνομα `Orismata.java`, και το μεταγλωττίσουμε, μπορούμε να το εκτελέσουμε δίνοντάς του *τουλάχιστον* τρία ορίσματα, για παράδειγμα με χρήση της εντολής:

```
java Orismata μια ωραία πεταλούδα
```

Το αποτέλεσμα της εκτέλεσης του προγράμματος `Orismata` στην περίπτωση αυτή θα είναι:

```
μια
ωραία
πεταλούδα
```

Σε όλες τις περιπτώσεις, η *σημασιολογία* (*semantics*) του προγράμματος είναι η ίδια: Η εκτέλεση των εντολών γίνεται ακολουθιακά (η μία μετά την άλλη), αρχίζοντας από την πρώτη και τελειώνοντας στην τελευταία εντολή της μεθόδου `main`.

Ασκήσεις 1.2.

1. Ποια από τα παρακάτω αναγνωριστικά είναι ορθά; Εξηγήστε γιατί τα υπόλοιπα δεν είναι.

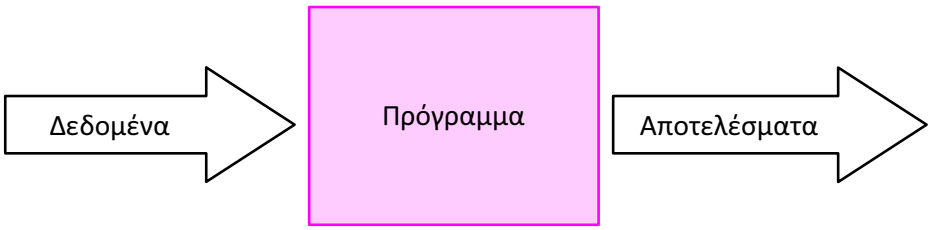
(α) `omega` **(β)** `prog.2` **(γ)** `5Bez710K`

(δ) `NaCl` **(ε)** `H2SO4` **(στ)** `Backus-Naur`

(ζ) `εφ(θ)` **(η)** `{` **(θ)** `Jaws 3`

1.5. Δεδομένα

Τα **δεδομένα** (**data**) παριστάνουν *πληροφορίες* που πρέπει να δοθούν σε ένα πρόγραμμα. Το πρόγραμμα επεξεργάζεται τις πληροφορίες αυτές για να παραγάγει αποτελέσματα, όπως φαίνεται στο Σχήμα 1.4. Τα δεδομένα μπορεί να προέρχονται από τον «εξωτερικό κόσμο» (συσκευές εισόδου), από αρχεία που βρίσκονται αποθηκευμένα στη βοηθητική μνήμη (συσκευές δίσκων) του υπολογιστή, ή από το Διαδίκτυο, και να αντιπροσωπεύουν γεγονότα, παρατηρήσεις, μετρήσεις, ιδέες για τον κόσμο κ.ά. Τα ζωντανά όντα αντλούν τα δεδομένα από εικόνες, κείμενα, ομιλίες κ.ά. μέσω των αισθήσεών τους[5]. Οι ηλεκτρονικοί υπολογιστές μπορούν και αυτοί να επεξεργαστούν, εκτός από κείμενα, δεδομένα ήχου, εικόνων κ.ά., εφόσον τα τελευταία έχουν ψηφιοποιηθεί (digitized).



Σχήμα 1.4. Πρόγραμμα.

Τα δεδομένα έχουν κάποιον **τύπο** (**type**) που ορίζει τη σημασία τους, δηλαδή το σύνολο των τιμών (values) που μπορούν να πάρουν και τις πράξεις που μπορούν να εκτελεστούν στις τιμές αυτές. Η Java υποστηρίζει οκτώ **πρωταρχικούς τύπους** (**primitive types**)[6], που παρουσιάζονται στον Πίνακα 1.1.

Οι τιμές των τύπων αυτών ανήκουν σε (γνήσια) υποσύνολα των *χαρακτήρων* της κωδικοποίησης UTF-16 (βλ. Εν. 4.1), της αλήθειας `(true)` και του ψέματος `(false)` (βλ. Κεφ. 3), των *ακεραίων* σε αναπαράσταση συμπληρώματος ως προς 2 (βλ. Εν. 2.1), και των *πραγματικών αριθμών* σε αναπαράσταση IEEE 754 (βλ. Κεφ. 6.1), αντίστοιχα. Σε αντίθεση με άλλες γλώσσες προγραμματισμού, το *μορφότυπο* (*format*) και το *μέγεθος* (*size*) των πρωταρχικών τύπων δεδομένων της Java είναι ρητά προκαθορισμένα και δεν εξαρτώνται από το κάθε σύστημα υπολογιστή, στο οποίο αυτή υλοποιείται. Γι' αυτόν το λόγο, τα προγράμματα που είναι γραμμένα σε Java είναι *φορητά* (*portable*). Τέλος, η Java μας επιτρέπει να ορίσουμε και *δικούς μας τύπους*.

[5] Τα δεδομένα δεν αντιπροσωπεύουν ακριβώς το φυσικό κόσμο, αλλά παρέχουν απλώς προσεγγιστικά μοντέλα τμημάτων του.

[6] Που είναι γνωστοί και ως *εσωτερικοί* ή *ενυπάρχοντες τύποι* (*intrinsics*).

Τύπος	Μέγεθος
`boolean`	1 δυφιοσυλλαβή
`byte`	1 δυφιοσυλλαβή
`char`	2 δυφιοσυλλαβές
`short`	2 δυφιοσυλλαβές
`int`	4 δυφιοσυλλαβές
`long`	8 δυφιοσυλλαβές
`float`	4 δυφιοσυλλαβές
`double`	8 δυφιοσυλλαβές

Πίνακας 1.1. Πρωταρχικοί τύποι της Java.

Μερικά από τα δεδομένα που χρησιμοποιούνται σε ένα πρόγραμμα δεν αλλάζουν ποτέ. Τα δεδομένα αυτά μπορούν να παρασταθούν με **σταθερές** (**constants**), οι οποίες είναι γνωστές και ως **κυριολεκτήματα** ή **κυριολεκτικές τιμές** (**literals**). Για να ορίσουμε στην Java μια σταθερά, γράφουμε την τιμή της. Για παράδειγμα, τα:

```
'_' (χαρακτήρας υπογράμμισης)   '@' (παπάκι)   '#' (δίεση)
false  true
0  -23  +478  32767
2200000000L  -897L  +1234561
3.14159265368f  -1.666E-14F  και
3.14159265358D  1.666E-14  -9.87654e+17  -0.1
```

ορίζουν σταθερές των τύπων `char`, `boolean`, `int`, `long`, `float` και `double`, αντίστοιχα.

Άλλα δεδομένα υπόκεινται σε αλλαγή. Για παράδειγμα, σε ένα πρόγραμμα ή σε έναν υπολογιστή τσέπης που προσθέτει αριθμούς, το τρέχον άθροισμα μεταβάλλεται κάθε φορά που προστίθεται σε αυτό ένας νέος αριθμός. Τα δεδομένα της μορφής αυτής παριστάνονται με *μεταβλητές*. Οι **μεταβλητές** (**variables**), όπως και τα πλήκτρα μνήμης των υπολογιστών τσέπης, παριστάνουν *θέσεις της κύριας μνήμης του υπολογιστή* (βλ. Εν. 1.1). Έτσι, όταν θα αναφερόμαστε στην «*τιμή μιας μεταβλητής*», θα εννοούμε στην πραγματικότητα *«τα περιεχόμενα της θέσης της μνήμης που αντιστοιχεί στη μεταβλητή αυτή»*. Σε αντίθεση με τους υπολογιστές τσέπης, οι υπολογιστές μάς επιτρέπουν να χρησιμοποιήσουμε όσες μεταβλητές χρειαζόμαστε, και να τις ονομάσουμε όπως μας βολεύει (δεν μας περιορίζουν σε ονόματα όπως `M`, `STO1` κ.λπ. που χρησιμοποιούν οι υπολογιστές τσέπης).

Η μεταβλητή δημιουργείται κατά την εκτέλεση του προγράμματος με τη δήλωσή της. Η **δήλωση μιας μεταβλητής** (**variable declaration**) πρέπει να περιλαμβάνει τον *τύπο* και το *αναγνωριστικό* της.

*Κατά σύμβαση, τα **αναγνωριστικά μεταβλητών** της Java είναι ουσιαστικά. Το πρώτο γράμμα ενός τέτοιου αναγνωριστικού είναι πεζό, ενώ το πρώτο γράμμα κάθε λέξης που περιέχεται σε αυτό (με εξαίρεση την πρώτη) είναι κεφαλαίο. Όλα τα υπόλοιπα γράμματα του αναγνωριστικού είναι πεζά, με εξαίρεση την περίπτωση των ακρωνυμίων που μπορεί να περιέχονται μέσα στο αναγνωριστικό και που γράφονται με κεφαλαία.*

Σχετικά παραδείγματα δηλώσεων μεταβλητών είναι τα εξής:

```
int wresErgasias, yperwries;
float aktina, emvadon;
char shmeioStikshs;
boolean petyxe;
double perifereia;
```

Η δήλωση μιας μεταβλητής αντιστοιχίζει απλώς το αναγνωριστικό της στη διεύθυνση μιας θέσης της μνήμης του υπολογιστή, και περιγράφει τον τύπο των τιμών που μπορούν να αποθηκευτούν στη θέση αυτή.

1.6. Παραστάσεις εκχώρησης, τοπικές και τελικές μεταβλητές

Μια **παράσταση εκχώρησης** (**assignment expression**) έχει *σύνταξη*:

```
αναγνωριστικό = παράσταση
```

όπου ο τύπος για την `παράσταση` που βρίσκεται στο δεξιό σκέλος του ειδικού συμβόλου = πρέπει να είναι ίδιος ή «βραχύτερος» σε σχέση με τον τύπο της μεταβλητής που βρίσκεται στο αριστερό σκέλος. Η σημασιολογία μιας παράστασης εκχώρησης είναι: *Υπολόγισε την τιμή της παράστασης που βρίσκεται δεξιά από το = και αποθήκευσέ τη (εκχώρησέ τη) στη θέση (διεύθυνση) που προσδιορίζει η μεταβλητή αριστερά από το =*. Σχετικά παραδείγματα είναι τα ακόλουθα (το σύμβολο * παριστάνει τον τελεστή «επί»):

```
wresErgasias = 8;
yperwries = 5;
aktina = 3.5f;
```

```
perifereia = 2 * 3.14159265368979323846 * aktina;
shmeioStikshs = '!';
petyxe = true;
```

Η αποτίμηση μιας παράστασης εκχώρησης έχει αποτέλεσμα την αλλαγή της τιμής της μεταβλητής που βρίσκεται στο αριστερό σκέλος της παράστασης. Η τιμή που είχε η μεταβλητή αυτή πριν από την αποτίμηση της παράστασης εκχώρησης χάνεται για πάντα. Οι τιμές των μεταβλητών που βρίσκονται στο δεξιό σκέλος της παράστασης εκχώρησης δεν αλλάζουν (εκτός και αν κάποια από αυτές βρίσκεται και στο αριστερό σκέλος της παράστασης).

Οι *μεταβλητές που δηλώνονται μέσα στο σώμα μιας μεθόδου* ονομάζονται **τοπικές μεταβλητές** (**local variables**) της μεθόδου αυτής. Όλες οι δηλώσεις των (τοπικών) μεταβλητών μιας μεθόδου ακολουθούν μετά το πρώτο `{` της μεθόδου. Όταν δημιουργείται μια τοπική μεταβλητή (μιας μεθόδου), μέσω της δήλωσής της, η τιμή της είναι *απροσδιόριστη* (*undefined*), εκτός και αν ο προγραμματιστής της δώσει ταυτόχρονα αρχική τιμή, όπως θα δούμε σε λίγο.

Αν θέλουμε ταυτόχρονα να δηλώσουμε μια μεταβλητή *και* να της δώσουμε αρχική τιμή, θα πρέπει να χρησιμοποιήσουμε μια παράσταση που περιλαμβάνει μια δήλωση *και* μια εκχώρηση. Σχετικά παραδείγματα τέτοιων παραστάσεων είναι τα εξής:

```
int wresErgasias = 8, yperwries = 5;
float aktina = 3.5f, emvadon;
double perifereia = 2*3.14159265368979323846 * aktina;
char shmeioStikshs = '!';
boolean petyxe = true;
```

Όλες οι μεταβλητές μιας παράστασης εκχώρησης πρέπει να έχουν δηλωθεί. Όλες οι μεταβλητές που βρίσκονται στο δεξιό σκέλος μιας παράστασης εκχώρησης πρέπει οπωσδήποτε να έχουν ήδη πάρει (ορισμένες) τιμές.

Η τιμή μιας παράστασης εκχώρησης είναι πάντα *η νέα τιμή της μεταβλητής που βρίσκεται στο αριστερό σκέλος της παράστασης*. Έτσι, η παράσταση:

```
wresErgasias = yperwries = 8;
```

έχει αποτέλεσμα την εκχώρηση της τιμής `8` στη μεταβλητή `yperwries` και κατόπιν την εκχώρηση της νέας τιμής της μεταβλητής `yperwries` (`8`) στη μεταβλητή `wresErgasias`.

Οι μεταβλητές που *διατηρούν την τιμή τους σταθερή* καθόλη τη διάρκεια εκτέλεσης του προγράμματος ονομάζονται **τελικές μεταβλητές** (**final variables**) στην Java, και στις δηλώσεις τους εμφανίζεται η λέξη-κλειδί `final` μπροστά από τον τύπο τους. Για παράδειγμα, οι παραστάσεις:

```
final int EVDOMADES_XRONOY = 52;
final char ΑΣΤΕΡΑΚΙ = '*';
final boolean ALHTHEIA = true;
final double Π = 3.14159265368979323846;
```

δηλώνουν και εκχωρούν τιμές σε τέσσερις τελικές μεταβλητές τύπου `int`, `char`, `boolean` και `double`, αντίστοιχα. Η δήλωση μιας τελικής μεταβλητής δίνει ένα όνομα (αναγνωριστικό) σε μια τιμή.

*Κατά σύμβαση, τα **αναγνωριστικά τελικών μεταβλητών** της Java είναι ουσιαστικά. Όλα τα γράμματα ενός τέτοιου αναγνωριστικού είναι κεφαλαία, ενώ οι επιμέρους λέξεις που περιέχονται μέσα σε αυτό χωρίζονται μεταξύ τους με _.*

Δεν επιτρέπεται να επιχειρήσουμε να τροποποιήσουμε μια τελική μεταβλητή. Τυχόν προσπάθεια να κάνουμε κάτι τέτοιο προκαλεί σφάλμα μεταγλώττισης.

Συμπερασματικά:

- Ο τύπος μιας μεταβλητής καθορίζει το σύνολο των τιμών που αυτή μπορεί να πάρει και τις πράξεις που μπορούν να εφαρμοστούν σε αυτή.
- Κάθε δεδομένο έχει έναν και μόνο ορισμένο τύπο.
- Ο τύπος μιας μεταβλητής εξαρτάται από τη δήλωσή της, και όχι από τις τιμές που μπορεί να πάρει κατά τη διάρκεια της εκτέλεσης του προγράμματος.
- Μια παράσταση μπορεί να είναι μια *σταθερά*, μια *μεταβλητή*, μια *κλήση μεθόδου*, ή ένας *συνδυασμός* σταθερών, μεταβλητών, και κλήσεων μεθόδων μέσω κατάλληλων τελεστών.

Προγράμματα 1.

1. Γράψτε ένα πρόγραμμα που να εμφανίζει τα παρακάτω σχήματα, το ένα δίπλα στο άλλο. Αλλάξτε το πρόγραμμά σας, ώστε να εμφανίζει τα σχήματα το ένα πάνω από το άλλο. Μήπως η ευκολία της αλλαγής του προγράμματός σας εξαρτάται από τον τρόπο με τον οποίο το έχετε γράψει;

```
          *                       *   *   *   *   *
        *   *                     *   *   *   *   *
      *   *   *                   *   *   *   *   *
    *   *   *   *                 *   *   *   *   *
  *   *   *   *   *               *   *   *   *   *
```

2. Γράψτε ένα πρόγραμμα που να εμφανίζει τα αρχικά σας με κεφαλαία γράμματα. Για παράδειγμα, αν το πρόγραμμα γραφόταν από το Γιάννη, θα έπρεπε να εμφανίζει:

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ
Java

Το βιβλίο απευθύνεται σε αναγνώστες που έχουν μικρή (ή ακόμα και μηδενική) πείρα στον προγραμματισμό και αποτελεί λεπτομερή εισαγωγή σε μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού. Java χρησιμοποιείται σήμερα, μεταξύ άλλων, για την ανάπτυξη διαδικτυακών και παραθυρικών εφαρμογών, εφαρμογών για κινητά τηλέφωνα και ενσωματωμένες συσκευές, καθώς και εφαρμογών για έξυπνες κάρτες. Ανεξάρτητα από το σκοπό για τον οποίο θέλετε να χρησιμοποιήσετε τη γλώσσα, στο βιβλίο αυτό θα βρείτε όλες τις βασικές γνώσεις που θα σας χρειαστούν για να ξεκινήσετε.
Πιο συγκεκριμένα, το βιβλίο καλύπτει:

- Τους πρωταρχικούς τύπους δεδομένων και τις βασικές δομές ελέγχου
- Τάξεις, αντικείμενα, πακέτα, κληρονομικότητα, πολυμορφισμό, διεπαφές, απαριθμητούς και φωλιασμένους τύπους
- Εξαιρέσεις, αρχεία, ρεύματα, κανάλια και ενταμιευτές
- Γενικότητες και συλλογές (λίστες, ουρές, στοίβες, σύνολα, απεικονίσεις).

Κάθε κεφάλαιο περιέχει παραδείγματα και ασκήσεις που μπορείτε να τροποποιήσετε, να μεταγλωττίσετε και να εκτελέσετε ώστε να κατανοήσετε καλύτερα τις έννοιες που παρουσιάζονται.

Ο **Γιάννης Κάβουρας** (1947-2011) ήταν Καθηγητής στο Τμήμα Πληροφορικής του Οικονομικού Πανεπιστημίου Αθηνών (Ο.Π.Α.). Σπούδασε Φυσικές Επιστήμες στο Πανεπιστήμιο Αθηνών και κατείχε τους μεταπτυχιακούς τίτλους Diploma in Computing Science (University of Glasgow), Master of Science (University of London, ICS) και Ph.D. (University of Glasgow) στην Επιστήμη Υπολογιστών. Εργάστηκε ως Lecturer στο Glasgow College of Technology (τώρα Glasgow Caledonian University), στο Heriot-Watt University του Εδιμβούργου, και στο University of Glasgow, καθώς και ως Επιστήμων Ερευνητής στο Ε.Κ.Ε.Φ.Ε. «Δημόκριτος». Διετέλεσε Προϊστάμενος του Κέντρου υπολογιστών του Ο.Π.Α. και επανειλημμένα Πρόεδρος του Τμήματος Πληροφορικής του ίδιου πανεπιστημίου. Υπήρξε κριτής επιστημονικών άρθρων για την ACM και άλλους ερευνητικούς φορείς. Τα επιστημονικά του ενδιαφέροντα περιλάμβαναν τους τομείς του προγραμματισμού, των συστημάτων υπολογιστών, και των λειτουργικών συστημάτων.

Η **Κατερίνα Ρουκουνάκη** σπούδασε Πληροφορική στο Οικονομικό Πανεπιστήμιο Αθηνών και είναι κάτοχος Master of Science in Information and Telecommunication Technologies (Athens Information Technology). Τα ερευνητικά της ενδιαφέρονται εστιάζονται στις γλώσσες προγραμματισμού και στην ανάπτυξη λογισμικού.

εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ

Δομοκού 4, Στουδίου Λαρίσης, 10440 ΑΘΗΝΑ, Τηλ. 210-5237635
www.klidarithmos.gr info@klidarithmos.gr
www.facebook.com/klidarithmos.gr

ISBN 978-960-461-464-6